15th International Command and Control Research and Technology Symposium (ICCRTS 2010)

ICCRTS Theme:	The Evolution of Command and Control
Track:	Information Sharing and Collaboration Processes and Behaviors (Track #3)
Title of Paper:	Enabling Tactical Edge Mashups with Live Objects
Authors:	Daniel Freedman (Cornell University, Department of Computer Science)
	Ken Birman (Cornell University, Department of Computer Science)
	Krzysztof Ostrowski (Cornell University, Department of Computer Science)
	Mark Linderman (Air Force Research Laboratory, Information Directorate)
	Robert Hillman (Air Force Research Laboratory, Information Directorate)
	Albert Frantz (Air Force Research Laboratory, Information Directorate)
Point of Contact:	Daniel Freedman
Office Address:	Cornell University
	Department of Computer Science
	Upson Hall
	Ithaca, NY 14853
E-mail Address:	dfreedman@cs.cornell.edu
Classification:	Distribution A: Approved for Public Release; distribution unlimited.
	(WPAFB PA # 88ABW-2010-2052)

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **JUN 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE **Enabling Tactical Edge Mashups with Live Objects**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Research Laboratory,Information Directorate,Rome,NY,13441**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Proceedings of the 15th International Command and Control Research and Technology Symposium (ICCRTS '10), Santa Monica, CA, June 22-24, 2010

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**16**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

(This page intentionally left blank.)

Enabling Tactical Edge Mashups with Live Objects

Daniel Freedman[1], Ken Birman[1], Krzysztof Ostrowski[1],
Mark Linderman[2], Robert Hillman[2], Albert Frantz[2]

[1] Department of Computer Science, Cornell University, Ithaca, NY, USA
[2] Information Directorate, Air Force Research Laboratory, Rome, NY, USA

We introduce the Live Objects framework, which leverages our distributed object-oriented programming model and enables tactical edge mashups for battlefield command and control. Unlike most deployed web services, which are typically limited to client-server interactions, Live Objects can simultaneously support multiple patterns of communication, including direct client-to-client protocols. This means that when clients are forward deployed or accessible only through disadvantaged links, a Live Objects system can remain highly responsive, whereas more standard solutions might slow down precipitously, become unresponsive, or fail outright. Here, we summarize the approach and then suggest that, when using it, a new kind of Service-Oriented Collaboration (SOC) application can be created that will combine direct client-to-client sharing of imagery, videos, or other real-time data captured in the field, with service-hosted data, including geographic information systems, weather prediction systems, social networks, and other databases. The client-to-client solutions can include powerful new collaboration features implemented to help the user achieve tactical Command and Control (C2) goals that require split-second coordination, for which reach-back to a server might be impossibly slow. We showcase key properties of our platform through a functional, proof-of-concept Combat Search and Rescue (CSAR) scenario.

I. Introduction

There is a growing opportunity within information systems to use *Service-Oriented Collaboration (SOC) Applications* in ways that can increase individual worker productivity, revolutionize medical informatics and decrease preventable healthcare mistakes, spur solutions in knowledge management, and otherwise transform spheres of collaboration and communication. Similar benefits within the military sector, and specifically for tactical Command and Control (C2) systems, are easily recognized in terms of soldier productivity, property book accountability, operations (S3) and intelligence (S2) planning and coordination, and tactical mission operations. This work discusses a representative joint military mission, namely Combat Search and Rescue (CSAR), as an example of the implementation of Live Objects and SOC applications within this field.

Collaborative applications will need to combine two types of content: traditional web service *hosted content*, such as data from geographic and topologic map servers, image repositories, personnel records, and other databases, with a variety of *collaboration features*, such as chat windows, white boards, peer-to-peer video and other media streams, and replication and coordination mechanisms.

Existing web service technologies make it easy to build applications in which *all data travels through a data center*. Implementing collaboration features using these technologies is problematic because collaborative applications can generate high, bursty update rates and yet often require low latencies and tight synchronization between collaborating users. One can often achieve better performance using direct client-to-client (also called peer-to-peer, or P2P) communication, but in today's Service-Oriented Architecture (SOA) platforms, "side-band" communication is difficult to integrate with hosted content. A growing number of publications on the integration of web services with peer-to-peer platforms (e.g., [2], [4], [9], [11], [12], [16], [17], [18], [22], [23]) reflect this challenge; the issue yet remains unresolved (see Section VI for more details).

As summarized in the abstract, we see this as a real issue in military settings and other kinds of systems that depart from the canonical wired network model on which Cloud Computing providers have focused. The forward-deployed unit or the soldier in the field will often have excellent connectivity to others nearby but very limited reach-back options, with appreciable loss rates, high latency, and perhaps low bandwidth. Thus, solutions that force data from the field to be transmitted through a centralized server are inappropriate. Instead, we would seek to leverage peer-to-peer protocols while at the same time benefitting from existing powerful server-hosted technologies.

Cornell's Live Distributed Objects platform [14] (Live Objects for short) allows even a non-programmer to construct content-rich solutions that blend traditional web services and peer-to-peer technologies, and to share them with others. The developer uses a drag-and-drop approach much like creating a slide show, after which the solution can be shared and used on other machines by non-developers. The users are immersed in the resulting collaborative application; they can interact with the application, and peers can see the results instantly. Updates are applied to all replicas in a consistent manner. Moreover, P2P communication can *coexist* with more standard solutions that reach back to hosted content and trigger updates at associated data centers. However, when an application needs high data rates, low latency, or special security, it can use protocols that bypass the data center to achieve the full performance of the network.

This paper makes the following contributions:

- We describe a new class of Service-Oriented Collaboration applications that integrate service-hosted content with peer-to-peer message streams. We analyze the CSAR scenario and list the key challenges that these kinds of applications place on their runtime environments.
- We describe a new class of multi-layered mashups and contrast them with more traditional, minibrowser-based approaches that are characteristic of today's web development.
- We discuss the advantages of decoupling transport and information layers as a means of achieving reusability, customizability, and rapid deployment and adaptation of collaboration applications in new environments. We discuss the resulting object-oriented perspective, in which instances of distributed communication protocols are modeled uniformly as objects similar to those in Java, .NET, Component Object Model (COM), or Smalltalk. We present our Live Objects platform as an example of a technology that fits well with the layered, componentized model derived through our analysis.
- We compare performance of hosted Enterprise Service Bus (ESB) solutions with peer-to-peer communication protocols as underlying communication substrates for SOC applications. While we do not find an obvious winner, we identify relative strengths of each of these solutions. We see this as further justification for the decoupling of information and transport layers advocated above and achieved in Live Objects: instead of a "one-size fits all" approach, an application can select among a menu of interchangeable components specialized for different environments.

II. Limitations of the Existing Model

There are two important reasons why integrating peer-to-peer collaboration with server-hosted content is difficult. The first is not strictly limited to collaboration and peer-to-peer protocols; rather, it is a general weakness of the current web mashup technologies that makes it hard to seamlessly integrate data from several different sources. The web development community has slowly converged towards service platforms that export autonomous interactive components to their clients, in the form of what we term *minibrowser* interfaces. A minibrowser is an interactive web page with embedded script, developed using AJAX, Silverlight, Caja, or similar technology, and optimized for displaying a single type of content, for example, interactive maps from Google Earth or Microsoft Virtual Earth.

The embedded script is often tightly integrated with back-end services in the data center, making it awkward to access the underlying services directly from a different script or a standalone client. As a result,

the only way such services can be mashed up with other web content is by either having the data center compute the mashup (so that it can be accessed via the minibrowser), or by embedding the entire minibrowser window in a web page. But an embedded minibrowser cannot seamlessly blend with the surrounding content; it functions instead as a standalone browser within its own frame and operates independently of the remainder of the web page.

To illustrate this point, examine Figures 1 and 2. These figures are screenshots of web applications, with content from multiple sources "mashed-up" together. Figure 1 is constructed using a standard web services approach, pulling content from Yahoo! maps and weather web services and assembling it into a web page as a set of tiled frames. Each frame is a minibrowser with its own interactive controls, and comes from a single content source. To illustrate one of the many restrictions, we note that the distinct frames are not synchronized; if the user pans or zooms in the map frame, the associated map will shift or zoom, but the other frames remain unchanged.

In Figure 2, we observe a similar application that leverages our Live Objects model. In this case, content from different sources is overlaid in the same window and synchronized. There are no frame boundaries, and elements of this mashup (including maps, planes, buildings, weather, chat, data, etc.) coexist as layers within which the end user can easily navigate. Data can come from many sources: this example actually overlays weather from Google on terrain maps from Microsoft's Virtual Earth platform, extracts census data from the US Census Bureau, and includes collaborative chat windows.

The second problem for integration of P2P with server-hosted content is that, with the traditional style of web development, content is assumed to be fetched from a server, either directly over Hyper Text Transport Protocol (HTTP), or by interacting with a web service. Web pages downloaded by clients' browsers contain embedded addresses of specific servers. Client-to-client traffic routes through a data center. So pervasive is this model that the web's HTTP is often viewed as the Internet Protocol (IP) of the next generation of applications [6], with IP relegated to a "low level" role. But HTTP, of course, is a client-server protocol.

In contrast, Live Objects allows visual content and update events to be communicated using any sort of protocol, not only client-server, but also overlay multicast, peer-to-peer replication, or even a custom domain-specific protocol designed by the content provider. These protocols could also provide coordination or other kinds of tactical roles in support of a mission. As noted earlier, this makes it possible to support critical services such as image or video sharing with extremely high levels of throughput and very low latency. It also enhances security as centralized servers need not see or process data exchanged directly between peers. Thus, the trusted domain is minimized, which eases assurance of proper security.

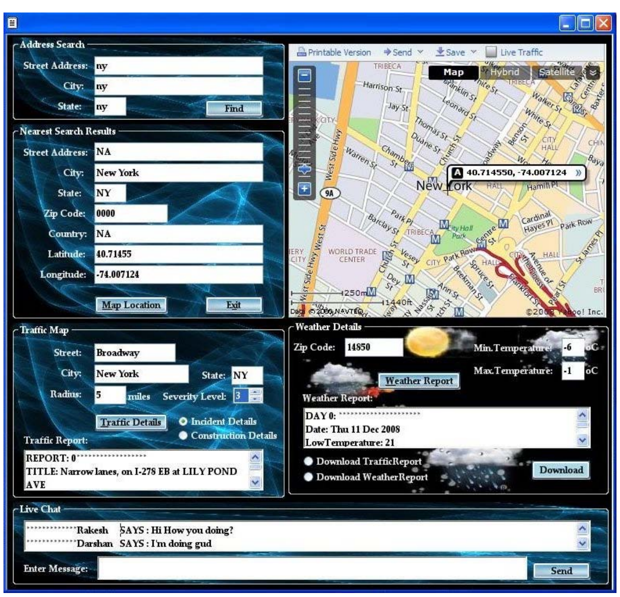

Figure 1: Standard Minibrowser-Style Mashup

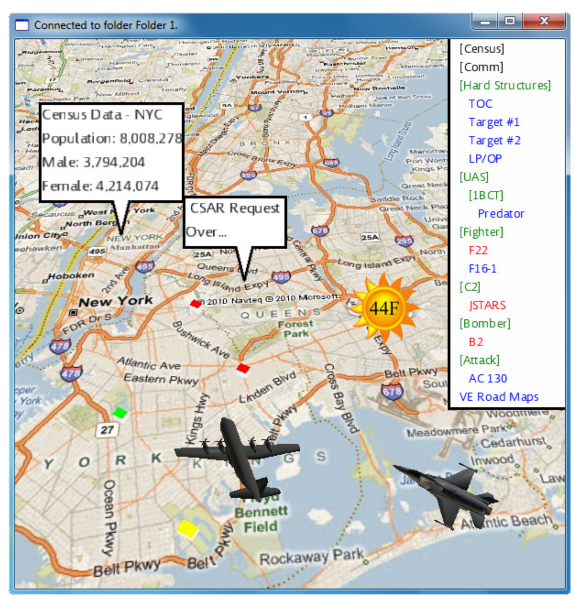

Figure 2: Live Objects Multi-Layered Mashup

The above discussion motivates our problem statement:

- Allow web applications to overlay content from multiple sources in a layered fashion, such that the distinct content layers share a single view and remain well synchronized: zooming, rotating, or panning should cause all layers to respond simultaneously, and an update in any of the layers should be reflected in all other layers.
- Allow updates to be carried by the protocol best matched to the setting in which the application is used.

As noted earlier, the solutions discussed here are based on Live Objects, which supports drag-and-drop application development. Of course, new types of components must be created for each type of content, but the existing collection of components provides access to several different types of web service hosted content (including all the examples given above). Once constructed, the resulting live application is stored as an eXtensible Markup Language (XML) file. The file can be moved about and even embedded in email. Users that open this file (using the associated local Live Objects runtime) find themselves immersed into the application. Nonetheless, as technologies such as Microsoft's Silverlight or Google's Wave advance, there is no reason that the same sorts of capabilities should not eventually become available through those platforms as well: today, they are rigid by design, not by necessity.

Examples of transport protocols optimized for various settings include support for networks with: network address translators (NATs) and firewalls (Self Organizing Live Objects, or SOLO [7]), low latency (Ricochet [1]), high throughput and very large numbers of nodes (QuickSilver Multicast, or QSM [13]), large numbers of irregularly overlapping multicast groups (Gossip Objects, or GO [3]), and strong reliability properties. Given modern power and performance trends, it is also very important that the technology leverage multicore parallelism [15].

III. Service Oriented Collaboration: CSAR Scenario

Before describing Live Objects in more detail, we first analyze an example collaboration application to expose the full range of needs and issues that arise.

Consider a military Combat Search and Rescue (CSAR) mission: large numbers of individual military members and units would likely contribute, spanning a spectrum of joint and coalition operations including ground, air, and potentially naval assets. The setting for the scenario on which we shall focus is the urban Area of Operations (AO) within Baghdad, Iraq, on the Western bank of the Tigris River; the season is Winter, 2004, with the first modern Iraqi elections scheduled to occur in scant months in January, 2005. The First Cavalry Division (1CD) maintains responsibility for this particular AO, liaising with Marine units on the opposite side of the Tigris. Among many such routine patrols of this particular day, a vehicle convoy from 4/5 Air Defense Artillery (ADA) of 1CD begins its daily hunt for insurgents emplacing Improvised Explosive Devices (IEDs); the platoon leader heading this particular convoy has already electronically filed his operations order, risk assessment, convoy route (including phase lines and checkpoints), and personnel, vehicle, and equipment rosters with the 4/5 ADA Tactical Operations Center (TOC). The Live Objects-enabled TOC has already autonomously integrated this data with existing hosted sources: Geographic Information System (GIS) overlays, weather forecasts, and demographic and cultural metadata; further, it has automatically instantiated Live Objects channels for future expected data feeds as the mission unfolds.

Soon after this patrol begins, an Air Force F-16, flying Combat Air Patrol (CAP) over Baghdad and responding to reports of sporadic mortar fire from atop a mosque, develops an unforeseen engine compressor stall at a relatively low altitude (AGL), outside of the range for an unpowered landing at Baghdad International Airport (BIAP). The pilot chooses to eject, but first transmits an emergency distress call to a circling Airborne Warning and Control Systems (AWACS) aircraft. The E-3 Sentry aircraft inputs this emergency call into its onboard Live Objects instance, which immediately integrates the last known state of the F-16 (including fuel levels,

armament status, exact Global Positioning System [GPS] coordinates, and pertinent airframe and electronics diagnostics) into the Fighter Wing's operation center, dispatches an operational-immediate call for CSAR teams to be dispatched to the projected grid, interrogates the USAF personnel files for pilot distress codes and the Army TOC for an updated Intelligence Preparation of Battlefield (IPB) to determine expected nearby insurgent forces, and otherwise disseminates this event throughout the military hierarchy. Live Objects channels thus carry emergency notifications, both horizontally (to joint forces and neighboring units) and vertically (up the chain-of-command) through the military organizational structure. Furthermore, such messages transit both at the edge of the physical, logical, and organizational network (in P2P fashion), as well as between the edge and the core (hosted services), as enabled by our framework.

Meanwhile, USAF CSAR assets take to the sky, in the form of a team of battle-tested pararescuemen (PJs or Pedros) aboard a Pavehawk helicopter; their crew chief expects to receive a steady flow of audio and visual Situation Reports (SITREPs) while airborne, via his Live Objects-enabled IP radio, to better vector the bird to the site of the downed pilot and to prepare for any potential resistance nearby. Simultaneously, our initial Army 4/5 ADA convoy is re-tasked by its battalion TOC to cordon off the reported crash site of the plane (separate from the location of the downed pilot); the convoy leader's handheld computer authenticates itself on the Live Object multicast channel associated with this contingency, joins the appropriate Publish-Subscribe (PubSub) groups pertaining to its role, and consequently both consumes and generates relevant data. As our convoy successfully contains the crashed F-16, Army and Naval Explosive Ordnance Disposal (EOD) units arrive onsite to handle the unexploded ordnance (UXO) from the hardpoints of the F-16's wings, so as to ensure that members of the local civilian population are not harmed by them. Meanwhile, throughout this controlled demolition process, the TOC has patched in leaders in Army Psychological Operations (PSYOPs) to ensure that Iraqi civilian leaders are aware of the crash and the efforts undertaken to protect civilian life.

Finally, the CSAR PJs hoist the pilot to safety, communicating this extraction to the on-scene assets as well as the centralized TOC and then immediately using Live Objects to stream vital signs from the pilot to the nearest Combat Support Hospital (CSH) so as to prepare the team of nurses and doctors for their imminent arrival. In all, Live Objects ensured immediate, resilient, and pervasive collaboration as the operation unfolded.

Having defined the scenario, we now analyze the requirements that it places on our tactical C2 tool. First, we emphasize the diversity of sources upon which this C2 application relies. Realistically, such sources would likely span multiple commands, and even services, within the Department of Defense; thus, tactical information for ground, sea, and air units are generated by their corresponding entities, as are topographic, weather, sensor, and demographic information, as well as alert traffic, ad infinitum. Data from distinct sources would generally require individual interfaces to account for independent protocols and unique formats.

Second, as conditions evolve, the team (TOC, S-2, J-3, etc., as appropriate) might need to modify the application, for instance, by adding new types or sources of information, changing the way data is represented, or even altering the communication protocols among team members (for example, if reach-back network links fail). Whereas a minibrowser would typically be prebuilt with all the available features in place, our scenario demands a much more flexible kind of tool that can be *redesigned* while in use.

Third, depending on the location and other factors, the best networking protocols and connectivity options may vary. In our CSAR scenario, the soldiers and airmen may have to use secure wireless P2P protocols (IP over UHF/VHF/FM) much of the time, reaching back to hosted services only intermittently when an unmanned aerial system (UAS) passes within radio range. More broadly, the right choice of protocol should reflect the operating conditions, and as these change, the platform should be capable of changing to a different protocol without disrupting the end user. This argues for a decoupling of functionality. Whereas a minibrowser packages the entire application into one object, our design better distinguishes the presentation object from objects representing information sources and objects representing transport protocols. Decoupling makes it possible to dynamically modify or even replace a component with some other (compatible) option when changing conditions require it.

We have posed what may sound like a very specialized scenario, but in fact we see this as a good example of a more general need arising in many kinds of military and tactical settings. For example, consider an Assistant S-3 who needs to deconflict naval gunfire with the axis of advance of his cavalry troop, while also integrating joint air missions into the mix. The mixture of operations information, geographic information system data, signal operating instructions, weather reporting, etc., may be just as rich and dynamic as in our search and rescue scenario, and the underlying communication options equally heterogeneous and unpredictable. Alternatively, a little farther afield, imagine the ability and efficiency gains for a hypothetical Inspector General (IG) audit that employed Live Objects to collect and collate the location (combination of hand-receipt and radio-frequency identification [RFID]), responsible party (company Property Book), condition (most recent DA-2404), and past unit history (chain of prior cyclic or sensitive inventories) for the every item in a given command's inventory, all correlated against desired end-state from the unit's Table of Organization and Equipment (TO&E) and tied into operational decision-making systems that judge a unit's effectiveness and ensure its capabilities to conduct a given mission. Certainly, rudimentary versions of such linked information systems currently exist in the form of web services, but the key operational design decision, forcing all communication through centralized servers, greatly curtails the effectiveness and resilience of the application. Similarly, a minibrowser pre-designed for a wired environment might perform poorly or fail outright under more dynamic conditions.

Thus, if we try to build this application using today's standard, off-the-shelf technologies such as Silverlight or Google Wave, we will likely construct a solution that cannot actually be deployed in the target environment. With Live Objects, we can construct our required mashup using a technology not limited to wired local-area network (LAN) and wide-area network (WAN) protocols with fast links back to data centers.

Throughout the above scenario, we noted a number of requirements; for clarity, we now summarize them below. As mentioned, these needs are seen in many settings and typical of most collaboration applications.

- We would like to enable a non-programmer to rapidly develop a new collaborative application by composing and customizing preexisting components.
- We would like to be able to overlay data from multiple sources, potentially in different formats, obtained using different protocols and inconsistent interfaces.
- We would like to be able to dynamically customize or modify the application at runtime, by incorporating new data sources or changing the way data is presented, without disrupting system operation.
- We would like to be able to accommodate new types of data sources, formats, and protocols that may not have been anticipated at the time the system was released.
- Data might be published by the individual users, and it might be necessary for the users to exchange their data without access to a centralized repository.
- Data may be obtained using different types of network protocols, and the type of the physical network or protocols may not be known in advance; it should be possible to rapidly compose the application using whatever communication infrastructure is currently available.
- Users may be mobile or temporarily disconnected, infrastructure may fail, and the topology of the network and its characteristics might change over time. The system should be easily reconfigurable.

The requirements outlined above might seem hard to satisfy, but, in fact, the solution is surprisingly simple. Our analysis motivates a component-oriented architecture, in which the web services and hosted content are modeled as reusable information layers backed by customizable transport channels, which together form a graph of components. A collaborative application is then a set of such graphs.

Our vision demands a new kind of collaboration standard, in order to facilitate the side-by-side coexistence of components that might today be implemented as proprietary minibrowsers: if we enable components to talk to one-another, we need to agree on the events and representation that the dialog will employ. The decoupling of functionality into layers also suggests a need for a standardized layering. In the scenario above,

one can identify at least four: the visualization layer, the linkage layer that talks to the underlying data source, the layer that generates and interprets updates, and the transport protocol). We propose that this decoupling be done using event-based interfaces (as originated in the Smalltalk language), which serve as a natural way to think about components.

Thus, rather than having the data-center developer offer content through proprietary minibrowser interface, the SOC application developer would define an event-based interface between transport and information layers; the visual events delivered by the transport could then be conveyed to an information layer responsible for visualizing them. This layer, in turn, would capture end-user mouse and keyboard input and pass such events down the network stack. With this type of event-based decoupling, any layer could easily be replaced with a different one.

In this perspective, distributed peer-to-peer protocols would also be encapsulated within their respective transport layers. Thus, for example, one version of a transport layer could fetch data directly from a server in a data center, whereas a different version might use a peer-to-peer dissemination architecture, such as a reliable multicast protocol; it could leverage different types of hardware or be optimized for different types of workloads. Provided that the different versions of the transport layer conform to the same standardized event-based interfaces, the application could then switch between them as conditions or users demand.

In this event-oriented world, end-users interact through Live Objects that transform actions into updates that are communicated in the form of events that are shared via the transport layer. The protocol implemented by the transport layer might replicate the event, deliver it to the visualization layer of our users, and report it through the event-based interface back to the information layer at which the event has originated. Thus, the transport layer with the embedded distributed protocol would behave very much like an object in Smalltalk: it would consume events and respond with events. This motivates thinking about communication protocols as objects, and indeed treating them much as we treat any other kind of object in a language like Java or in a runtime environment like Jini or .NET. Doing so unifies apparently distinct approaches. Just as remotely hosted forms of content, such as a map or image can be modeled as an object, so can network protocols be treated as objects.

Some P2P systems try to make everything a P2P interaction. But in the CSAR scenario examined above, several kinds of content would more naturally be hosted: topographic maps and 3D images of terrain, buildings, and targets, and other Imagery Intelligence (IMINT) sources, weather information, enemy Order of Battle (OOB), etc. On the other hand, collaboration applications are likely to embody quite a range of P2P event streams: each separate video object, GPS source, sensor, etc., may have its own associated update stream. If one thinks of these as topics in publish-subscribe (PubSub) eventing systems, an application could have many such topics, and the application instance running on a given user's machine could simultaneously display data from several topics. We have previously said that we would like to think of protocols as objects. It now becomes clear that further precision is needed: the objects are not merely protocols, but in fact are individual protocol instances. Our system will need to simultaneously support potentially large numbers of transport objects running concurrently in the end-user's system, in support of a variety of applications and uses.

All of this leads to new challenges. The obvious one was mentioned earlier: today's web services do not support P2P communication. Contemporary web service solutions presume a client-server style of interaction, with data relayed through a message-oriented middleware broker. Even though individual clients may remain connected to one another, if they lose connectivity to this central server which functions as the middleware broker, they will no longer be able to collaborate.

Another serious issue arises if the clients do not trust the data center: sensitive data will need to be encrypted. The problem here is that web service security standards tend to trust the web services platform itself. The standards offer no help at all if we need to provide end-to-end encryption mechanisms while also preventing the hosted services from observing the raw data and workloads.

Finally, we encounter debilitating latency and throughput issues: hosted services will be performance-limiting bottlenecks when used in settings with large numbers of clients, as we show below in Section V.

We summarize both the positive and negative perspectives of existing client-server applications:

- 👍 Web services standardize client access to hosted services and data: we can easily build some form of multi-framed web page that could host each kind of information in its own minibrowser.
- 👍 When connectivity is adequate, relaying data via a hosted service has many of the benefits of a PubSub architecture, such as robustness as the set of clients changes.
- 👎 The natural way to think of our application is as an object-oriented mashup, but web services provide no support for this kind of client application development.
- 👎 The solution may perform very poorly, or fail if the hosted services are inaccessible.
- 👎 All data will probably be visible to the hosted services unless the developer uses some sort of non-standard end-to-end cryptography.

IV. Using Live Objects for Collaboration

Our Live Objects platform supports componentized, layered mashup creation and sharing, thus overcoming many limitations of existing web technologies. We have used Live Objects to construct a number of SOC applications, some of which are quite sophisticated, including a solution to the CSAR scenario as formulated in Section III. The major design aspects are as follows:

- The developer starts by creating (or gaining access to) a collection of components. Each component is an object that supports live functionality and exposes event-based interfaces by which it interacts with other components. Examples include:
  - Components representing hosted content
  - Sensors and actuators
  - Renderers that graphically depict events
  - Replication protocols
  - Synchronization protocols
  - Folders containing sets of objects
  - Display interfaces that visualize folders
- Mashups of components are represented as a kind of XML web pages, each describing a "recipe" for obtaining and parameterizing components that will serve as layers of the composed mashup. We term such an XML page a *live object reference*. References can be distributed as files, via email, through HTTP, or by other means.
- The application is created by building a forest consisting of graphs of references that are mashed together. At design time, an automated tool lets the developer drag and drop to combine references for individual objects into an XML mashup of references describing a graph of objects.
- The platform type-checks mashups to verify that they compose correctly. For example, a 3-D visualization of an airplane may need to be connected to a source of GPS and other orientation data, which in turn needs to run over a data replication protocol with specific reliability, ordering, or security properties.
- When activated on a user's machine, an XML mashup yields a graph of interconnected *proxies.* A proxy is a piece of running code that may render, decode, or transform visual content, encapsulate a protocol stack, and so on. Each sub-component in the XML mashup produces an associated proxy. The hierarchy of proxies reflects the hierarchical structure of the XML mashup.
- If needed, an object proxy can initialize itself by copying the state from some active proxy (our platform assists with this sort of *state transfer*).

- The object proxies then become active ("live") by relaying events from other objects into a replication channel or by receiving events and reacting to them (for example, by redisplaying an aircraft).

Our approach shares certain similarities with the existing web development model, in that it uses hierarchical XML documents to define the content. On the other hand, we depart from some of the de-facto stylistic standards that have emerged. For example, if one pulls a minibrowser from Google Earth, this minibrowser expects to interact directly with the end user and includes embedded JavaScript that handles such interactions. In Live Objects, the same functionality would be represented as a mashup of a component that fetches maps and similar content with a second component that provides the visualization interface.

Although the term *mashup* may sound static, in the sense of having its components predetermined, this is not necessarily the case. One kind of live object could be a folder including a set of objects, for example extracted from a directory in a file system or pulled from a database in response to a query. When the folder contents change, the mashup is dynamically updated, as might occur when an army squad enters a building or changes direction at a street intersection during Military Operations on Urban Terrain (MOUT) warfare.

Thus, Live Objects can easily support applications that dynamically recompute the set of "visible" objects, as a function of location and orientation, and dynamically add or remove them from the mashup. A rescuer would automatically and instantly be shown the symbols of others who are already working at that position and be able to follow team SITREPs or point-to-point dialog with them, through chat objects that run over multicast protocol objects. This model can support a wide variety of collaboration and coordination paradigms.

In summary, the Live Objects platform makes it easy for a non-programmer to create the needed application. In our CSAR scenario, the TOC pulls prebuilt object references from a folder, each corresponding to a desired kind of information. Hosted data, such as weather, terrain maps, etc., would correspond to objects that "point" to a web service over the network. Peer-to-peer objects would implement chat windows, soldier Liquid / Ammo / Casualty / Equipment (LACE) status updates, squad video feeds, etc. Event interfaces allow such objects to coexist in a shared display window that can pan, zoom, jump to new locations, etc.

The relative advantages and disadvantages of our model can be summarized as follows:

- 👍 Like other modern web development tools, our platform supports drag-and-drop style of development, permitting fast, easy creation of content-rich mashups.
- 👍 The resulting solutions are easy to share.
- 👍 By selecting appropriate transport layers, functionality such as coordination between searchers can remain active even if connectivity to the data center is disrupted.
- 👍 Streams of video or sensor data can travel directly and will not be delayed by the need to "ricochet" off a remote, and potentially inaccessible, server.
- 👎 New event-based interoperability standards are needed. Lacking them, we could lose access to some of the sophisticated proprietary interactive functionality optimized for proprietary minibrowser-based solutions with embedded JavaScript.
- 👎 Direct peer-to-peer communication can be much harder to use than relaying data through a hosted service that uses an ESB model. Furthermore, the lack of a "one size fits all" PubSub substrate forces the developers to become familiar with and choose between a range of different and incompatible options. An inappropriate choice of transport could result in degraded Quality of Service (QoS), inferior scalability, or even data loss.

V. Performance Evaluation

Central to our argument is the assertion that hosted event notification solutions scale poorly and stand as a barrier to collaboration applications, and that developers will want to combine hosted content with P2P protocols to overcome these problems. In this section we present data to support our claims. Some of the

results (Figure 3 and Figure 4) are drawn from a widely cited industry whitepaper by Fiorano Software [8] and were obtained using a testing methodology and setup developed and published by Sonic Software [20]. The remaining measurements derive from our own experiments.

In Figure 3, Fiorano Software analyzes the performance of several commercial ESB products, plotting the maximum throughput (in messages per second) for 1024-Byte messages. The experiment varies the number of subscribers while using a single publisher that communicates through a single hosted message broker on a single topic. Brokers are configured for message durability: even if a subscriber experiences a transient loss of connectivity, the publisher retains and hence can replay all messages. As the number of subscribers increases, performance degrades sharply. Although not shown, latency will also soar because the amount of time the broker needs to spend sending a single message increases linearly with the number of subscribers.

In collaboration applications, durability is often not required. Figure 4 examines throughput in an experiment in which the publisher does not log data, in which a disconnected subscriber would experience a loss. We find that while the maximum throughput is much higher, the degradation of performance is even more dramatic. One could improve scalability using clustered service structures, but such a step would have no impact on latency, since clients would still need to relay data through the data center. We note that hosted ESB solutions do not necessarily scale well, and that the client-to-data center communication path could introduce intolerable performance overheads.

Next, we report on experiments conducted independently at Cornell University, focusing on scalability of event notification platforms that leverage peer-to-peer techniques for dissemination and recovery. In Figure 5, we compare the maximum throughput of two decentralized reliable multicast protocols, again with 1024-byte messages, a single topic, and a single publisher. Unlike in the previous external Fiorano tests, which utilized a gigabit network, these experiments use a 100 Mbps LAN, limiting the peak performance to 10,000 messages per second. QSM [13] achieves stable high throughput (saturating the network). JGroups, a popular product, runs at about a fifth of that speed, collapsing as the number of subscribers increases. Also, at small loss rates, latency in QSM is at the level of 10-15 ms irrespectively of the number of subscribers.

Figure 6 demonstrates that QSM maintains its high performance when the number of topics is varied. Here, we report performance for 110 subscribers within QSM, but performance for other group sizes is similar. We examine JGroups performance, for configurations that vary between 2 and 110 subscribers, as a function of number of topics. While we observe higher JGroups performance with smaller group sizes, this erodes as the number of topics increases. JGroups failed when we attempted to configure it with more than 256 topics.

Finally, we look at two scalable protocols under conditions of “stress,” with a focus on delivery latency (y-axis) as a fixed message rate is spread over varying numbers of topics. Sixty-four subscribers each join some number of topics; a publisher sends data at a rate of 1000 messages per second, selecting the topic in which to send at random. Our experimental setup, on Emulab, injects a random 1% message loss rate. In Figure 7, we see that Ricochet [1], a Cornell-developed protocol for low-latency multicast, maintains steady low-latency delivery (about 10 ms; logarithmic y-axis) as the number of topics increases to 1024 (logarithmic x-axis). In contrast, latency soars when we repeat this with the industry-standard Scalable Reliable Multicast (SRM), widely used for event notification in commercial datacenters. As can be seen in the graph, SRM’s recovery latency rises linearly in the number of topics, reaching almost eight seconds with 128 groups.

To summarize, our experiments confirm that:

- Hosted enterprise service bus architectures can achieve high levels of PubSub performance for small numbers of subscribers, but performance degrades very sharply as the number of subscribers or topics grows.
- JGroups and SRM, which do not leverage P2P techniques, scale poorly in the number of subscribers or topics. QSM and Ricochet, where subscribers cooperate, scale well in these dimensions.
- Ricochet achieved the best recovery latency when message loss is an issue (but at relatively high overhead, not shown on these graphs). At small loss rates, QSM achieves similar average latency with

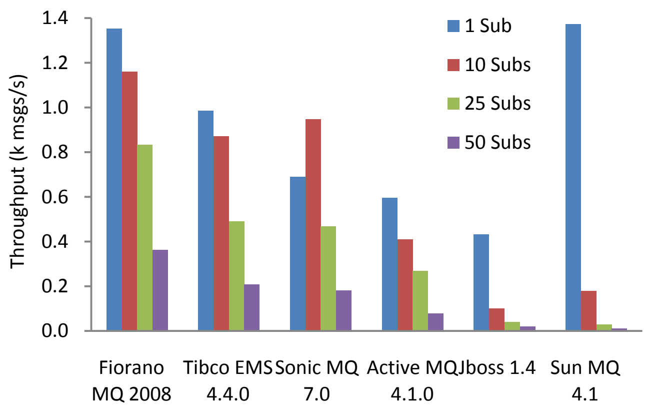

Figure 3: Scalability of commercials ESBs

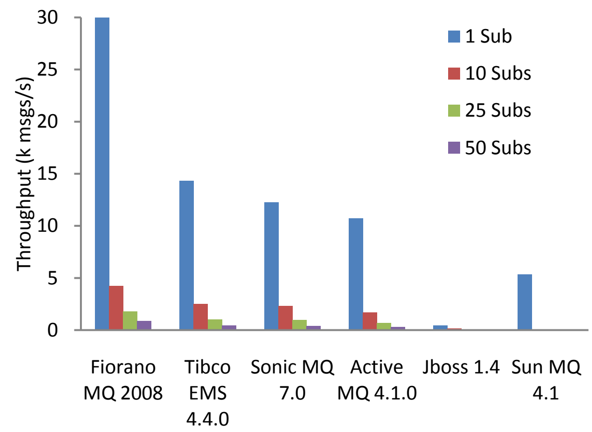

Figure 4: Scalability of commercials ESBs (without data logging)

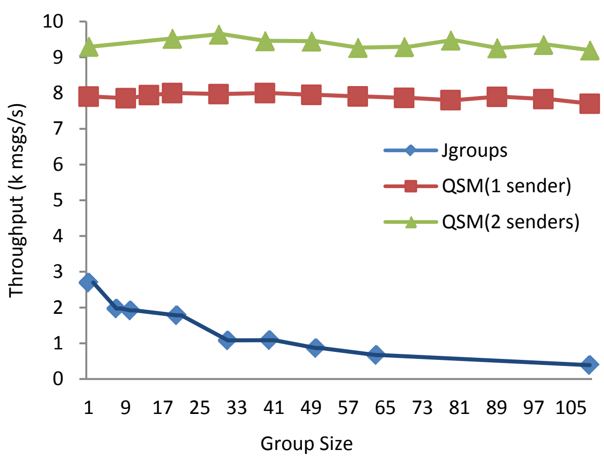

Figure 5: Scalability of QSM and JGroups (throughput for various group sizes)

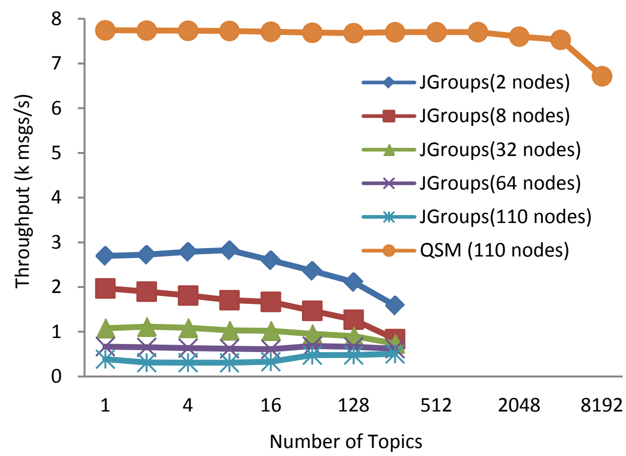

Figure 6: Scalability of QSM and JGroups (throughput for various numbers of topics)

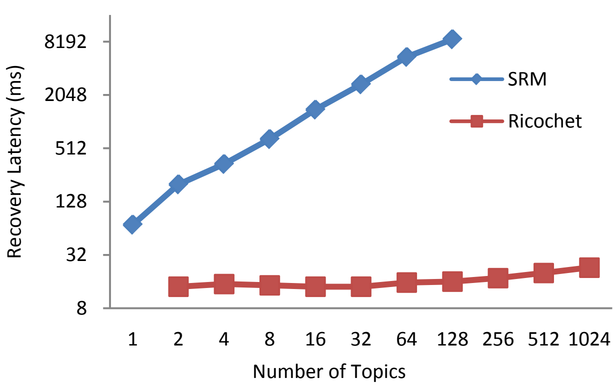

Figure 7: Delivery latency for SRM and Ricochet with varying numbers of topics

considerably lower network overheads, but, if a packet is lost, it may take several seconds to recover it, making it less appropriate for time-critical applications.

We do not see any single optimal choice here: each of the solutions tested has some advantages that its competitors lack. Indeed, we are currently developing a new P2P protocol suite, called SOLO [7]; it builds an overlay multicast tree within which events travel, and is capable of self-organizing in the presence of firewalls, NATs, and bottleneck links. A separate project involves a new protocol suite known as the Properties Framework [15]. The goal is to offer strong forms of reliability that can be customized for special needs. Thus, speed and scalability are only elements of a broader story. Developers will need different solutions for different purposes.

VI. Prior Work

The idea of integrating web services with peer-to-peer platforms is certainly not new ([2], [4], [9], [11], [12], [16], [17], [18], [22], [23]). The existing work falls roughly into two categories. The first line of research is focused on the use of peer-to-peer technologies, particularly Juxtapose (JXTA), as a basis for scalable web service discovery. The second line of research concentrates on the use of replication protocols at the web service back-end to achieve fault-tolerance. In both cases, P2P platforms such as JXTA are treated not as means of collaboration or media carrying live content, but rather as a supporting infrastructure at the data center backend. In contrast, our work is focused on blending the content available through P2P and web service protocols; neither technology is subordinate with respect to the other.

Technologies that use P2P protocols to support live and interactive content already exist; an excellent example of such technology is the Croquet [19] collaboration environment, in which the entire state of a virtual 3D world is stored in a peer-to-peer fashion and updated using a two-phase commit protocol. Other work in this direction includes the “Serverless 3D World” [21]. However, none of these systems supports the sort of componentized, layered architectures that we advocate here. The types of peer-to-peer protocols these systems can leverage, and the types of traditional hosted content they can blend with their P2P content, are limited. In contrast, our platform is architected with such extensibility in mind; every part of it can be replaced and customized, and different components within a single mashup application can leverage different transport protocols.

Prior work on typed component architectures includes a tremendous variety of programming languages and platforms, including early languages such as SmallTalk alongside modern component-based environments such as Java, .NET, or COM, specialized component architectures such as MIT’s Argus system [10] , flexible protocol composition stacks such as Bast [5], service-oriented architectures such as Juni, and others. None of these, however, has been used in the context of integrating service-hosted and peer-to-peer content. Discussion of component integration systems and their relation to Live Objects, however, is beyond the scope of this paper. Ostrowski and coauthors [14] provide more details.

Furthermore, much relevant prior work consists of the scripting languages mentioned in the discussion above: JavaScript, Caja, Silverlight, and others. As explained earlier, our belief is that even though these languages are intended for fairly general use, they have evolved to focus on minibrowser situations in which the application lives within a dedicated browser frame, interacts directly with the user, and cannot be mixed with content from other sources in a layered fashion. Live Objects can support minibrowsers as objects, but we argue that we gain flexibility by modeling hosted content at a lower level as components that interact via events and by focusing on the multi-layered style of mashups as opposed to the standard tiled model.

Finally, this paper builds extensively upon our previous discussion of the application of Live Objects to Service-Oriented Computing [24], with added detail and discussions that examine its use in tactical mashups, both generally as well as in the form of the specific CSAR scenario in Section III.

VII. Conclusions

To build ambitious collaboration applications, the web services community will need ways to combine (to "mash up") content from multiple sources. These include hosted sources that exist in data centers and support web service interfaces, but also direct peer-to-peer protocols capable of transporting audio, video, realtime chat and other content at high data rates with low latency. They must allow disconnected collaboration, without mandatory reach-back to data centers.

Our review of the performance of ESB eventing solutions in the standard hosted web services model clarifies that hosted event channels do not have the scalability and latency properties needed by many applications. P2P alternatives often achieve far better scalability, lower latency, and higher throughput. They also have security advantages in that the data center does not see, and thus potentially compromise, every event.

The Live Objects platform can seamlessly support applications that require a mixture of data sources, including both hosted and direct P2P event-stream data. Further benefits include an easy to use drag-and-drop programming style that yields applications represented as XML files, which can be shared as files or even via email. Users that open such files find themselves immersed in a media-rich collaborative environment that also offers strong reliability, high performance, impressive scalability and (in the near future) a powerful type-driven security mechanism. Most important of all, Live Objects are real: the platform is distributed under a FreeBSD open-source license through Microsoft's managed CodePlex source repository, as well as directly from Cornell University.

VIII. References

[1] Mahesh Balakrishnan, Ken Birman, Amar Phanishayee, Stefan Pleisch. "Ricochet: Lateral Error Correction for Time-Critical Multicast." NSDI 2007.

[2] Farnoush Banaei-Kashani, Ching-Chien Chen, Cyrus Shahabi. "WSPDS: Web Services Peer-to-peer Discovery Service." ICOMP 2004.

[3] Ken Birman, Anne-Marie Kermarrec, Krzysztof Ostrowski, Marin Bertier, Danny Dolev, Robbert van Renesse. "Exploiting Gossip for Self-Management in Scalable Event Notification Systems." DEPSA 2007.

[4] Jorge Cardoso. "Semantic integration of Web Services and Peer-to-Peer networks to achieve fault-tolerance." IEEE GrC 2006.

[5] Benoit Garbinato, Rachid Guerraoui. "Flexible Protocol Composition in Bast." ICDCS 1998.

[6] Kelly Jackson Higgins. "Microsoft Report: Physical Data Theft, Trojans Up; Bug Disclosure Down." *Security Dark Reading* (2008).

[7] Qi Huang, Ken Birman. "SOLO: Self Organizing Live Objects." Technical Report (December, 2008).

[8] "JMS Performance Comparison for Publish Subscribe Messaging". Fiorano Software Technologies Pvt. Ltd., February 2008.

[9] Timo Koskela, Janne Julkunen, Jari Korhonen, Meirong Liu, Mika Ylianttila. "Leveraging Collaboration of Peer-to-Peer and Web Services." UBICOMM 2008.

[10] Barbara Liskov and Robert Scheifler. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs." *TOPLAS* **5**, 381-404 (1983).

[11] Shenghua Liu, Peep Küngas, Mikhail Matskin. "Agent-based web service composition with JADE and JXTA." SWWS 2006.

[12] Federica Mandreoli, Antonio Perdichizzi, Wilma Penzo. "A P2P-based Architecture for SemanticWeb Service Automatic Composition." DEXA 2007.

[13] Krzysztof Ostrowski, Ken Birman, Danny Dolev. "QuickSilver Scalable Multicast (QSM)." NCA 2008.

[14] Krzysztof Ostrowski, Ken Birman, Danny Dolev, Jong Hoon Ahnn. "Programming with Live Distributed Objects." ECOOP 2008.

[15] Krzysztof Ostrowski, Chuck Sakoda, Ken Birman. "Self-Replicating Objects for Multicore Platforms." ECOOP 2010.

[16] Mike Papazoglou, Bernd Krämer, Jian Yang. "Leveraging Web-Services and Peer-to-Peer Networks." CAiSE 2003.

[17] Changtao Qu, Wolfgang Nejdl. "Interacting the Edutella / JXTA Peer-to-Peer Network with Web Services." SAINT 2004.

[18] Mario Schlosser, Michael Sintek, Stefan Decker, Wolfgang Nejdl. "A Scalable and Ontology-based P2P Infrastructure for Semantic Web Services." P2P 2002.

[19] David Smith, Alan Kay, Andreas Raab, David Reed. "Croquet: A Collaboration System Architecture." C5 2003.

[20] Sonic Performance test suite, available at:
http://www.sonicsoftware.com/products/sonicmq/performance_benchmark/index.asp

[21] Egemen Tanin, Aaron Harwood, Hanan Samet, Sarana Nutanong, Minh Tri Truong. "A Serverless 3D World." GIS 2004.

[22] Minjun Wang, Geoffrey Fox, Shrideep Pallickara. "A Demonstration of Collaborative Web Services and Peer-to-Peer Grids." ITCC 2004.

[23] Zhenqi Wang, Yuanyuan Hu. "A P2P Network Based Architecture for Web Service." WiCom 2007.

[24] Ken Birman, Jared Cantwell, Daniel Freedman, Qi Huang, Petko Nikolov, Krzysztof Ostrowski. "Building Collaboration Applications that Mix Web Services Hosted Content with P2P Protocols." ICWS 2009.